Présentation L-Py: L-Systems in Python

Présenté par Christophe Pradal dans Poster and Demo Session 2011

Résumé

L-Py: L-Systems in Python

Authors: Frederic Boudon (VirtualPlants team, CIRAD/INRIA Montpellier, France) and Christophe Pradal (VirtualPlants team, CIRAD/INRIA Montpellier, France)

Context

Lindenmayer-systems were conceived as a mathematical framework for modeling growth of plants. In this paper, we present L-Py, a simulation package that mixes L-systems construction with the Python high-level modeling language. In addition to this software module, an integrated visual development environment has been developed that facilitates the creation of plant models. In particular, easy to use optimization tools have been integrated. Thanks to Python and its modular approach, this framework makes it possible to integrate a variety of tools defined in different modeling context, in particular tools from the OpenAlea platform [Pradal08]. Additionally, it can be integrated as a simple growth simulation module into more complex computational pipelines.

LSystem in Python

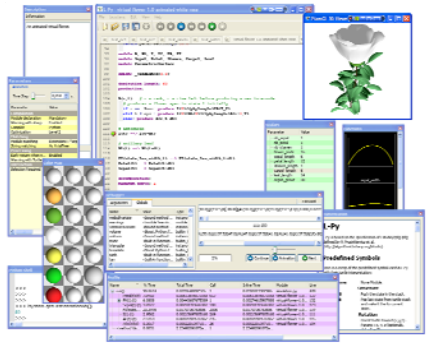
Formally, an L-systems is defined as a triplet made of an alphabet V of modules, an axiom which is a string of modules representing initial state of the structure and some production rules. The central idea of L-systems consists of rewriting the modules of a string in parallell with the production rules. Indeed, the rewriting rules (productions) express the creation and changes of state of the plant modules throughout time. Such rules can be expressed using a dedicated programming language [Prusinkiewicz99] or by incorporating L-system-based language constructs into existing languages, such as C++ [Karwowski03] or Java [Kniemeyer08]. In this work, we explore the use of the Python language as support for L-systems. Finally, a 3D geometric representation is computed with a **turtle** interpretation of the string.

In our system, L-systems constructs are integrated into regular python code. In a first part of the L-Py code of a model, modules can optionally be declared. The axiom (initial string) is then declared. The keywords production initiates block for rules declaration. Rule predecessor includes specification of possible contexts that condition the application of a rule. Rules can be expressed using two conventions. Simple rules can be written in a compact mathematical style. For more complex rules, successor specifications are declared using the nproduce statement embedded into regular python code. A typical L-Py code will look like

```
1   from random import uniform                  # std python code
2   max_age, dr, branching_prob = 10, 0.01, 0.5   # constants
3   module Apex(age), Internode(length,radius)    # modules declaration
4   Axiom: Apex(0)
5   production:
6   Internode(l,r) --> Internode(l,r+dr)
7   Apex(age) :
8     if age < max_age:
9        nproduce Internode(1+random(),0.1)/(180)
10       if uniform(0,1) < branching_prob: nproduce [+(30) Apex(age+1) ]
11       nproduce Apex(age+1)
```

When an L-system is interpreted, its code, which contains both python code and special L-system constructs, is first transformed in pure python code by the L-Py language parser procedure. A simulation library, written in C++, can then execute the model. Its main job consists in parsing the string and checking for matching of modules of the string with rules predecessors. In case of matching, the Python functions corresponding to the rule code are called. This simulation kernel is available as a simple Python library using *Boost.Python*. Thus L-Py can be encapsulated as a simple component of a more complex pipeline which integrates other formalisms.

For the geometric interpretation of the model, a representation is computed using the 3D turtle interpretation of the string, with predefined modules encoding turtle actions [Prusinkiewicz90]. For this, L-Py makes use of the graphical library PlantGL [PB2009]. Indeed, a PlantGL scene graph is computed for each required geometrical representation.



L-system models can be complemented with a number of facilities to express high-level visual attributes such as functions, materials, etc. Upon the L-Py kernel, an integrated development environment has been built using **PyQt** that includes code and visual parameter editors. It contains a code editor with dedicated syntax highlighting. Visual editors make it possible to edit interactively 2D functions, scalars, curves, patches and materials, etc. A Python shell allows to inspect and manipulate procedurally the L-systems and their results. L-Py also contains a continuous modelling mode in which interactions with parameters are immediately propagated and visualization of the model automatically updated. This allows simple interaction with the model. To keep acceptable performance, L-Py integrates a debugger that gives the successive rule applications and a profiler, built upon *cProfile*, that shows the time spent in each rules and functions and makes it possible to identify bottlenecks in the execution.

Example of applications

Two main applications developed with L-Py will be presented. The first one is the MappleT model, which simulates the growth of a realistic apple tree during 4 years and integrating biomechanics for branch bending, stochastic models for branching pattern and physiological rules for organ development. Thus L-systems and Python with its scientific libraries provides a powerfull framework for plant modelling. The second application is the use of L-Py for education. An experiment with high school students of Montpellier has been made during one school year. It consists of learning turtle geometry and L-systems to create of a virtual scrubland of south of France. L-Py has been perceived intuitive enough to be used by students to produce a complex realistic botanical scenery.

Fig 1. Simulation of the growth of an apple tree during 4 years.

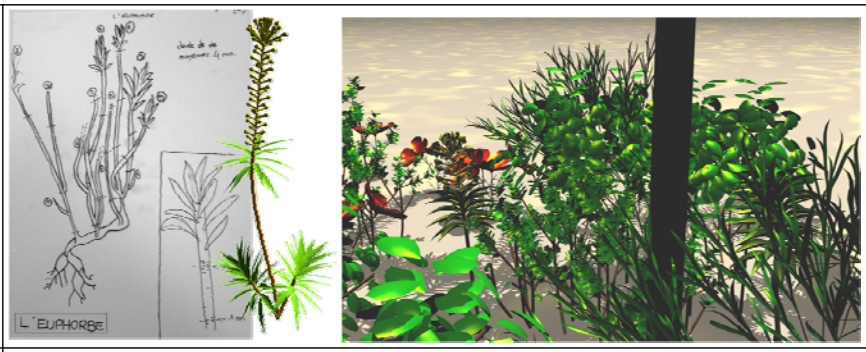


Fig 2. Virtual landscape created by high school students

References

[Pradal08] Pradal C, Dufour-Kowalski S, Boudon F, Fournier C, Godin C, 2008. OpenAlea: A visual programming and component-based software platform for plant modeling, Functional Plant Biology, 35 (9 & 10): 751-760.

[Prusinkiewicz99] Prusinkiewicz P, Hanan J, Mech R, 1999. An L-system-based plant modeling language. In: M. Nagl, A. Schuerr and M. Muench (Eds): Applications of graph transformations with industrial relevance. Proceedings of the International workshop AGTIVE'99, Lecture Notes in Computer Science 1779: 395-410, Springer, Berlin.

[Karwowski03] Karwowski R, Prusinkiewicz P, 2003. Design and implementation of the L+C modeling language. Electronic Notes in Theoretical Computer Science 86 (2).

[Kniemeyer08] Kniemeyer O, Kurth W, 2008. The modelling platform GroIMP and the programming language XL. In Applications of Graph Transformations with industrial Relevance: Third international Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and invited Papers, A. Schürr, M. Nagl, and A. Zündorf, Eds. Lecture Notes In Computer Science, Springer-Verlag, 5088: 570-572.

[Prusinkiewicz90] Prusinkiewicz P, Lindenmayer A, 1990. The Algorithmic Beauty of Plants. With J. Hanan, F. Fracchia, D. Fowler, A. de Boer and L. Mercer. Springer, New York.

[PB2009] Pradal C, Boudon F, Nouguier C, Chopard J, Godin C, 2009. PlantGL: A Python-based geometric library for 3D plant modelling at different scales, Graphical Models 71(1): 1-21.

présentation #4458 - dernière mise à jour 2011/05/26, créé le 2011/05/20 par Tiziano Zito